Protocol Analysis and Verification with the Cryptographic Protocol Shapes Analyzer (CPSA)
A Student Guide

Edward V. Zieglar Jr.
R22 - Secure Systems Architecture and Analysis
R2 -Laboratory for Advanced Cybersecurity Research
evziegl@nsa.ic.gov

January 14, 2021

This course guide serves to assist the student in understanding the need for formal analysis and verification of the security properties of cryptographic protocols and to instruct the student in the use of one such tool for formal analysis and verification, the Cryptographic Protocol Shapes Analyzer (CPSA). This guide is intended to be used in a classroom setting where students have access to an instructor, the CPSA manual, and machines capable of running the CPSA command line tools and a modern web browser such as Firefox, Safari, Chrome or Edge. Students are expected to have some basic knowledge of network protocols, cryptography, network security, programming, and command-line skills.

Part I

Introduction to Formal Protocol Analysis and Verification

Chapter 1

Why Use Formal Analysis and Verification

The following excerpt is taken from Chapter 11 of the second edition of *Network Security: Private Communication in a Public World* by C. Kaufman, R. Perlman and M. Speciner, published in 2002 [4].

Begin Excerpt from [4]

SECURITY HANDSHAKE PITFALLS

Knock Knock!
Who's there?
Alice.
Alice who?

and you'll have to read on to find secure ways of continuing...

Security in communications almost always includes an initial authentication handshake, and sometimes, in addition, integrity protection and/or encryption of the data. Let's assume Alice and Bob wish to communicate. In order to communicate, they need to know some information about themselves and about the other party. Some of this information is secret. Some usually isn't, such as the names Alice and Bob.

In 9.3 Cryptographic Authentication Protocols we described some example security handshakes. Although they may seem straightforward, minor variants of secure protocols can have security holes. As a matter of fact, many deployed protocols have been designed with security flaws.

This stuff just isn't that hard. How come nobody gets it right?
–Al Eldridge

This book does not tell you the one "best" protocol. Different protocols have different tradeoffs. Some threats are more likely in some situations, and different resources are available in terms of computational power, specialized hardware, money to pay off patent holders, humans willing and able to be careful, and so forth. We mortals should never need to design our own cryptographic algorithms (like DES, RSA, or MD5–we can leave that in the able hands of Ron Rivest and a handful of other specialists. But it is often the case that people outside the security community, such as implementers or protocol designers, have to design security features into protocols–including authentication handshakes. It is crucial that potential flaws be well understood. Even when a protocol is patterned after known protocols, the slightest alteration, even in something that "couldn't conceivably matter", can introduce subtle security flaws. Or even if a new flaw is not introduced, a weakness that was not important in the original protocol might be serious in a different environment.

In practice, the way that security protocols are designed is by starting with some design, which is almost certainly flawed, and then checking for all the weaknesses one can imagine, and fixing them. So the more educated we can become about the types of flaws likely to be in a protocol, the more likely it is that we'll understand all the properties of a protocol we deploy.

In this chapter we describe some typical protocols and evaluate them according to performance (number of messages, processing power required, compactness of messages) and security.

End Excerpt from [4]

This informal approach to the design of cryptographic protocols, as illustrated in the above passage from a network security textbook [4], of relying on rules of thumb, best practices, and the imagination of the designers has been taught for decades and continues to be taught. Papers such as "Using encryption for authentication in large networks of computers," [6] in 1978 and "Prudent engineering practice for cryptographic protocols," [1] in 1994 were originally written to teach developers the best practices and errors to avoid. The practices they outline have been included in many network security textbooks The question you should be asking yourself is are there better approaches?

Chapter 2

The Dolev-Yao Intruder Model

Whitfield Diffie and Martin Hellman introduced the world to Public Key Cryptography in 1976 with the publication of "New Directions in Cryptography." [2] Their paper describes what has become known as the Diffie-Hellman key exchange and was based on ideas that Ralph Merkle created to provide secure communication over an insecure channel as part of a class project in 1974 and later published [5] in 1978. The introduction of public key techniques led to the development of a large number of security protocols that employed such techniques.

Although the public key techniques used in the protocols were usually effective against passive eavesdroppers, an improperly designed protocol could be vulnerable to an active adversary, who may impersonate other users or modify or replay messages that are transmitted. Such protocols might be compromised in complex ways which would make informal arguments about their security prone to error. Dolev and Yao set out to describe a formal model in which the security issues could be described precisely. They describe their model in [3]. There model is based on the following basic assumptions, taken directly from [3] (note: E_x represents the public key of X and D_x represents the private key of X in a public key system):

---Begin Excerpt from [3]

1. In a perfect public key system,
   (a) the one-way functions used are unbreakable;
   (b) the public directory is secure and cannot be tampered with;
   (c) everyone has access to all E_x;
   (d) only X knows D_x.

2. In a two-party protocol, only the two users who wish to communicate are involved in the transmission process; the assistance of a third party in decryption or encryption is not needed.

3. In a uniform protocol, the same format is used by every pair of users that wish to communicate. In the three examples given previous, the user's names A, B are symbolic parameters and can be any two names.

4. With respect to the behavior of the saboteur, we will focus attention on saboteurs who are "active" eavesdroppers. That means someone who first taps the communication line to obtain messages and then tries everything he can in order to discover the plaintext. More precisely, we will assume the following about a saboteur:

   (a) He can obtain any message passing through the network.

   (b) He is a legitimate user of the network, and thus in particular can initiate a conversation with any other user.

   (c) He will have the opportunity to be a receiver to any user A. (More generally, we allow the possibility that any user B may become a receiver to any other user A.)

End Excerpt from [3]

2.1 Class Exercise

Meeting Mayhem Game

The objective of this game is for participants to arrange a meeting place and time by sending messages using a messaging application on the network while the adversary attempts to control the meeting place and time of the participants.

Chapter 3

Strand Spaces

Part II

Analyzing Protocols with CPSA

Chapter 4

Overview of CPSA

Chapter 5

Authentication

The following examples are taken from Chapter 11, "Security Handshake Pitfalls" in [4]. We will work through the examples from the text to introduce the concepts for modeling and analyzing cryptographic protocols with CPSA. The following four protocols are based on a single authentication as described in the section of text from [4] below.

Begin Excerpt from [4]

LOGIN ONLY

A lot of existing protocols were designed in an environment where eavesdropping was not a concern (rightly or wrongly), and bad guys were (rightly or wrongly) not expected to be very sophisticated. The authentication in such protocols generally consists of:

- Alice (the initiator) sends her name and password (in the clear) across the network to Bob.
- Bob verifies the name and password, and then communication occurs, with no further attention to security–no encryption, no cryptographic integrity protection.

A very common enhancement to such a protocol is to replace the transmission of the cleartext password with a cryptographic challenge/response. First we'll discuss protocols based on shared secrets, using either secret key cryptographic algorithms or message digest algorithms. Then we'll discuss similar protocols using public key technology.

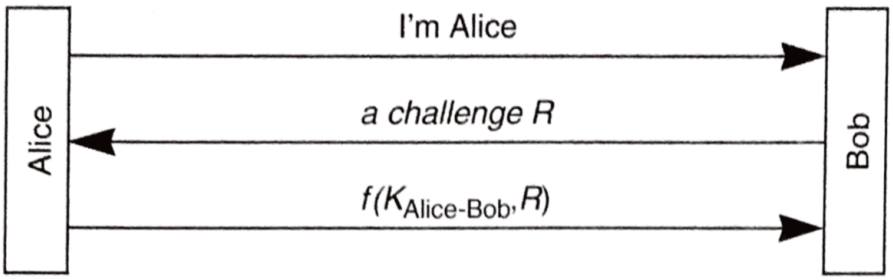
End Excerpt from [4]

5.1 One-way Shared Secret

Begin Excerpt from [4]

The notation $f(K_{Alice-Bob}, R)$ means that R is cryptographically transformed, somehow, with Alice and Bob's shared secret $K_{Alice-Bob}$. This could be done by using $K_{Alice-Bob}$ as a secret key in some algorithm such as DES or AES, and using $K_{Alice-Bob}$ to encrypt R. Or it could be done by hashing R and $K_{Alice-Bob}$, for instance by concatenating R and $K_{Alice-Bob}$ and computing a message digest on the result. When we explicitly mean encryption with $K_{Alice-Bob}$ we'll write $K_{Alice-Bob}\{R\}$. When we explicitly mean a hash, we'll write $h(K_{Alice-Bob}, R)$ or $hash(K_{Alice-Bob}, R)$.

Consider Protocol 11-1. An eavesdropper will see both R and $f(K_{Alice-Bob}, R)$. It is essential that seeing the pair does not enable the eavesdropper to derive $K_{Alice-Bob}$.



Protocol 11-1. Bob authenticates Alice based on a shared secret $K_{\text{Alice-Bob}}$

This protocol is a big improvement over passwords in the clear. An eavesdropper cannot impersonate Alice based on overhearing the exchange, since next time there will be a different challenge. However, there are some weaknesses to this protocol:

- Authentication is not mutual. Bob authenticates Alice, but Alice does not authenticate Bob. If Trudy can receive packets transmitted to Bob's network address, and respond with Bob's network address (or through other means convince Alice that Trudy's address is Bob's), then Alice will be fooled into assuming Trudy is Bob. Trudy doesn't need to know Alice's secret in order to impersonate Bob–she just needs to send any old number R to Alice and ignore Alice's response.
- If this is the entire protocol (i.e., the remainder of the conversation is transmitted without cryptographic protection), then Trudy can hijack the conversation after the initial exchange, assuming she can generate packets

with Alice's source address. It's also useful to Trudy, but not absolutely essential, that she be able to receive packets transmitted to Alice's network layer address. (See Homework Problem 1.)

- An eavesdropper could mount an off-line password-guessing attack (assuming $K_{Alice-Bob}$ is derived from a password), knowing R and $f(K_{Alice-Bob}, R)$. (Recall that an off-line password guessing attack is one in which an intruder captures information against which passwords can be tested in private, so in this context it means guessing a password, turning that password into a key K, and then seeing whether $f(K, R)$ equals $f(K_{Alice-Bob}, R)$.)

- Someone who reads the database at Bob can later impersonate Alice. In many cases it is difficult to protect the database at Bob. There might be many servers where Alice uses the same password, and although the administrators of most of the servers might be very conscientious about security (not letting unauthorized people get near their machines, and enforcing unguessable passwords), it only takes one unprotected server for an intruder to read the relevant information. Furthermore, protecting the database implies protecting all the backup media as well, by either preventing access to it (locking it in a safe) or encrypting the contents and somehow protecting the key with which it was encrypted.

Despite these drawbacks, if there are limited resources available for adding security, replacing the cleartext password transmission is the single most important security enhancement that can be done.

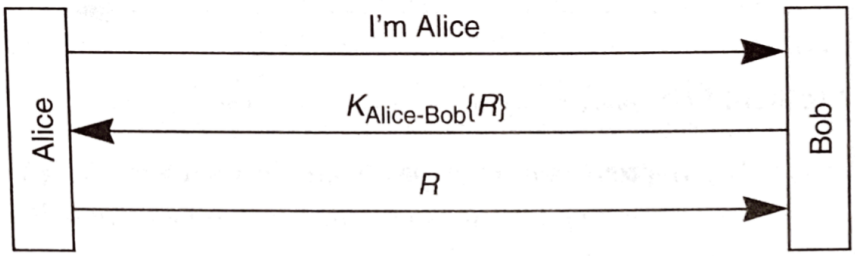
End Excerpt from [4]

To model this or any protocol, there are some questions that need to be answered:

1. What are the security goals of the protocol?
2. How many protocols need to be modeled?
3. How many roles exist in the protocol?
4. How many messages are there?
5. What variables will we need?
6. What functions will we need?
7. Which CPSA algebra should we use?
8. What are the assumptions being made?

Begin Excerpt from [4]

A minor variant on Protocol 11-1 is the following:



Protocol 11-2. Bob authenticates Alice based on a shared secret key $K_{\text{Alice-Bob}}$

In this protocol Bob chooses a random challenge R, encrypts it, and transmits the result. Alice then decrypts the received quantity, using the secret key $K_{Alice-Bob}$ to get R, and sends R to Bob. This protocol has only minor security differences from Protocol 11-1:

- This protocol requires reversible cryptography, for example a secret key cryptographic algorithm. Protocol 11-1 can be done using a hash function. For example, $f(K_{Alice-Bob}, R)$ could be the message digest of $K_{Alice-Bob}$ concatenated with R. But in Protocol 11-2, Alice has to be able to reverse what Bob has done to R in order to retrieve R. Sometimes there is a performance advantage to being able to use one of the message digest functions rather than having to use, say, DES. Sometimes there are export issues involved in having code for encryption available, even if it's only used for authentication, whereas using a message digest function would be less likely to create export problems.

- Suppose $K_{Alice-Bob}$ is derived from a password and therefore vulnerable to a dictionary attack. If R is a recognizable quantity, for instance a 32-bit random number padded with 32 zero bits to fill out an encryption block, then Trudy can, without eavesdropping, mount a dictionary attack by merely sending the message I am Alice and obtaining $K_{Alice-Bob}\{R\}$. If Trudy is eavesdropping, however, and sees both R and $K_{Alice-Bob}\{R\}$, she can mount a dictionary attack with either protocol. It is often the case that eavesdropping is more difficult than merely sending a message claiming to be Alice. Kerberos V4 (see Chapter 13 Kerberos V4) is an example of a protocol that has this security weakness.

- If R is a recognizable quantity with limited lifetime, such as a random number concatenated with a timestamp, Alice authenticates Bob because

only someone knowing $K_{Alice-Bob}$ could generate $K_{Alice-Bob}\{R\}$. To accomplish mutual authentication, R must be limited lifetime to foil the replaying of an old $K_{Alice-Bob}\{R\}$.

End Excerpt from [4]

5.1.1 Class Exercise

Specify Protocol 11-2, create skeletons to analyze the protocol, and run the CPSA command line tools to complete your analysis.

Hint: You can use the specification for protocol 11-1 as a starting point.

Answer these questions:

1. What are the security goals this protocol claims to provide?

2. What skeletons are required to validate those security goals?

3. Is the security provided by this protocol equivalent to protocol 11-1?

Begin Excerpt from [4]

Another variant on Protocol 11-1 is to shorten the handshake to a single message by having Alice use a timestamp instead of an R that Bob supplies:

Protocol 11-3. Bob authenticates Alice based on synchronized clocks and a shared secret $K_{Alice\text{-}Bob}$

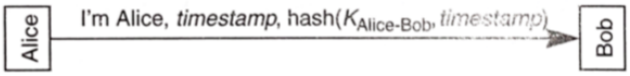
This modification requires that Bob and Alice have reasonably synchronized clocks. Alice encrypts the current time. Bob decrypts the result and makes sure the result is acceptable (i.e., within an acceptable clock skew). The implications of this modification are:

- This modification can be added very easily to a protocol designed for sending cleartext passwords, since it does not add any additional messages?it merely replaces the cleartext password field with the encrypted timestamp in the first message transmitted by Alice to Bob.
- The protocol is now more efficient. It goes beyond saving two messages. It means that a server, Bob, does not need to keep any volatile state (such as R in Protocol 11-1) regarding Alice (but see next bullet). This protocol can be added to a request/response protocol (such as RPC) by having Alice merely add the encrypted timestamp into her request. Bob can authenticate the request, generate a reply, and forget the whole thing ever happened.
- Someone eavesdropping can use Alice's transmitted $K_{Alice-Bob}\{timestamp\}$ to impersonate Alice, if done within the acceptable clock skew. This threat can be foiled if Bob remembers all timestamps sent by Alice until they ?expire? (i.e., they are old enough that the clock skew check would consider them invalid).
- Another potential security pitfall occurs if there are multiple servers for which Alice uses the same secret $K_{Alice-Bob}$. Then an eavesdropper who acts quickly can use the encrypted timestamp field Alice transmitted, and (if still within the acceptable time skew) impersonate Alice to a different server. This can be foiled by concatenating the server name in with the timestamp. Instead of sending $K_{Alice-Bob}\{timestamp\}$, Alice sends $K_{Alice-Bob}\{"Bob"|timestamp\}$. That quantity would not be accepted by a different server.
- If our bad guy Trudy can convince Bob to set his clock back, she can reuse encrypted timestamps she had overheard in what is now Bob's future. In practice there are systems that are vulnerable to an intruder resetting

the clock. Although it might be obvious that a password file would be something that needed to be protected, if the security protocols are not completely understood, it might not be obvious that clock-setting could be a serious security vulnerability.

- If security relies on time, then setting the time will be an operation that requires a security handshake. A handshake based on time will fail if the clocks are far apart. If there's a system with an incorrect time, then it will be impossible to log into the system in order to manage it (in order to correct its clock). A plausible solution to this is to have a different authentication handshake based on challenge/response (i.e., not dependent on time) for managing clock setting.

In Protocol 11-1, computing $f(K_{Alice-Bob}, R)$. may be done with a secret key encryption scheme using $K_{Alice-Bob}$ as a key, or by concatenating $K_{Alice-Bob}$ with R and doing a hash. When we're using timestamps the same is true (a message digest works), except for a minor complication. How does Bob verify that $hash(K_{Alice-Bob}, R)$ is reasonable? Suppose the timestamp is in units of minutes, and the believable clock skew is 10 minutes. Then Bob would have to compute $hash(K_{Alice-Bob}, timestamp)$ for each of the twenty possible valid timestamps to verify the value Alice sends (though he could stop as soon as he found a match). With a reversible encryption function, all he had to do was decrypt the quantity received and see if the result was acceptable. While checking twenty values might have acceptable performance, this approach would become intolerably inefficient if the clock granularity allows a lot more legal values within the clock skew. For instance, the timestamp might be in units of microseconds. There are 600 million valid timestamps within a five-minute clock skew. This would be unacceptably inefficient for Bob to verify. The solution (assuming you wanted to use a microsecond clock and a hash function rather than a reversible encryption scheme) is to have Alice transmit the actual timestamp encrypted, in addition to transmitting the hashed value. So the protocol would be:



Protocol 11-4. Bob authenticates Alice based on hashing a high-resolution time and a shared secret $K_{Alice-Bob}$

End Excerpt from [4]

5.1.2 Class Exercise

Specify Protocols 11-3 and 11-4, create skeletons to analyze the protocols, and run the CPSA command line tools to complete your analysis.

Answer these questions:

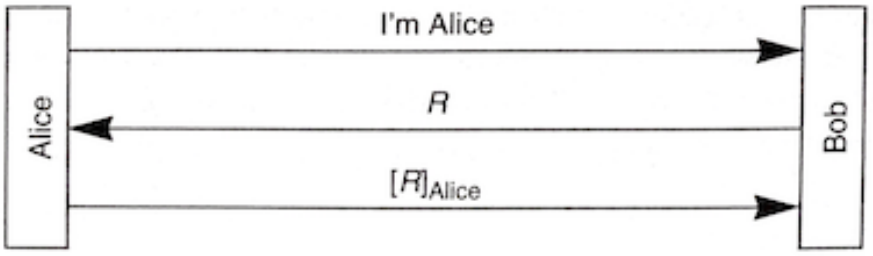
1. How did you model timestamps in CPSA?
2. Does your modeling of timestamps impact your analysis?
3. Do both protocols achieve the same security goals?
4. Can you demonstrate with CPSA the attacks that are described in text discussing the protocols?

5.2 One-Way Public Key

Begin Excerpt from [4]

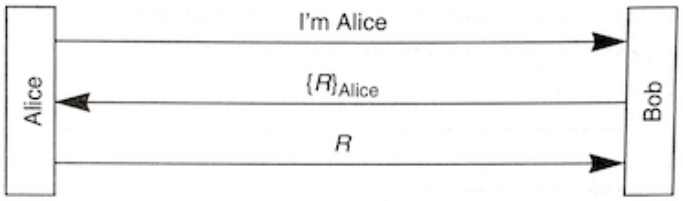
With protocols in the previous section, which are based on shared secrets, Trudy can impersonate Alice if she can read Bob's database. If the protocols are based on public key technology instead, this can be avoided, as in Protocol 11-5.

In this case $[R]_{Alice}$ means that Alice signs R (i.e. transforms R using her private key). Bob will verify Alice's signature $[R]_{Alice}$ using Alice's public key, and accept the login if the result matches R. This is very similar to Protocol 11-1. The advantage of this protocol is that the database at Bob is no longer security-sensitive to an attacker reading it. Bob's database must be protected from unauthorized modification, but not from unauthorized disclosure.



Protocol 11-5. Bob authenticates Alice based on her public key signature

And as before, the same minor variant works:



Protocol 11-6. Bob authenticates Alice if she can decrypt a message encrypted with her public key

In this variant, Bob chooses R, encrypts it using Alice's public key, and Alice proves she knows her private key by decrypting the received quantity to retrieve R. A problem with this variant is that some public key schemes (such as DSS) can only do signatures, not reversible encryption. So in those cases this variant cannot be used.

In both Protocol 11-5 and Protocol 11-6 there is a potential serious problem. In Protocol 11-5 you can trick someone into signing something. That means, if you have a quantity on which you'd like to forge Alice's signature, you might be able to impersonate Bob's network address, wait for Alice to try to log in, and then give her the quantity as the challenge. She'll sign it, and now you know her signature on that quantity. Protocol 11-6 has Alice decrypting something. So, if there's some encrypted message someone sent to Alice and you're wondering what's in it, you might again impersonate Bob's address, wait for Alice to log in, and then have Alice decrypt it for you.

How can we avoid getting in trouble? The general rule is that you should not use the same key for two different purposes unless the designs for all uses of the key are coordinated so that an attacker can't use one protocol to help break another. An example method of coordination is to ensure that R has some structure. For instance, if you sign different types of things (say an R in a challenge/response protocol versus an electronic mail message), each type of thing should have a structure so that it cannot be mistaken for another type of thing. For example, there might be a type field concatenated to the front of the quantity before signing, with different values for authentication challenge and mail message. Part of the purpose of the PKCS standards (see 6.3.6 Public-Key Cryptography Standard (PKCS)) is to impose enough structure to prevent this sort of problem when the same RSA key is used for different purposes.

Note the chilling implication–you can design several schemes where each is independently secure, but when you use more than one, you can have a problem. Perhaps even more chilling, you could design a new protocol whose deployment would compromise the security of existing schemes (if the new protocol used the same keys).

End Excerpt from [4]

5.2.1 Class Exercise

Specify Protocols 11-5 and 11-6, create skeletons to analyze the protocols, and run the CPSA command line tools to complete your analysis.

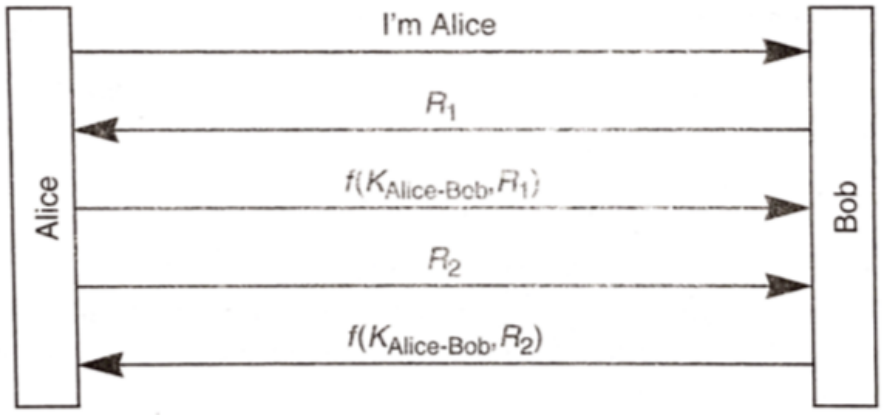
Answer these questions:

1. What security goals are these protocols intended to satisfy?
2. Does your CPSA analysis of your protocol models validate the security goals?
3. Do both protocols achieve the same security goals?
4. The description of the protocols describes a potential serious problem. Does CPSA indicate this potential problem? Why or why not?
5. What would you need to model to identify that potential problem?

5.3 Mutual Authentication

Begin Excerpt from [4]

Suppose we want to do mutual authentication, i.e. Alice will know for sure she is communicating with Bob. We could just do an authentication exchange in each direction:



Protocol 11-7. Mutual authentication based on a shared secret $K_{\text{Alice-Bob}}$

End Excerpt from [4]

5.3.1 Class Exercise

Specify Protocols 11-7, create skeletons to analyze the protocol, and run the CPSA command line tools to complete your analysis.

Answer these questions:

1. What security goals does the protocol intend to satisfy?
2. Does your CPSA analysis of your protocol model validate the security goals?
3. Did your CPSA analysis complete?
4. If your analysis does not indicate that the goals are satisfied, can you identify the problem?

Bibliography

[1] Abadi, M., and Needham, R., Prudent engineering practice for cryptographic protocols. In: IEEE Symposium on Research in Security and Privacy, pp. 122–136. IEEE Computer Society Press (1994).

[2] Diffie, Whitfield, and Hellman, Martin E., New Directions in Cryptography. IEEE Transactions on Information Theory. 22 (6), 644–654 (1976).

[3] Dolev, D., and Yao, A. C., On the Security of Public Key Protocols. IEEE Transactions on Information Theory. IT-29 (2), 198-208 (1983).

[4] Kaufman, C., Perlman, R., and Speciner, M., *Network Security: Private Communication in a Public World*, second edition, Prentice Hall PTR, Upper Saddle River, NJ, 2002.

[5] Merkle, Ralph C., Secure Communications over Insecure Channels. Communications of the ACM 21(4), 294–299 (1978).

[6] Needham, R., and Schroeder, M.D., Using encryption for authentication in large networks of computers. Communications of the ACM 21(12), 993–999 (1978).

[7] Rowe, P. D., Guttman, J. D., and Ramsdell, J. D., Assumption-Based Analysis of Distance-Bounding Protocols with CPSA. In: LNCS, pp. ??–??. (2020).